NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Programming Techniques for Supercomputers Tutorial

Erlangen National High Performance Computing Center

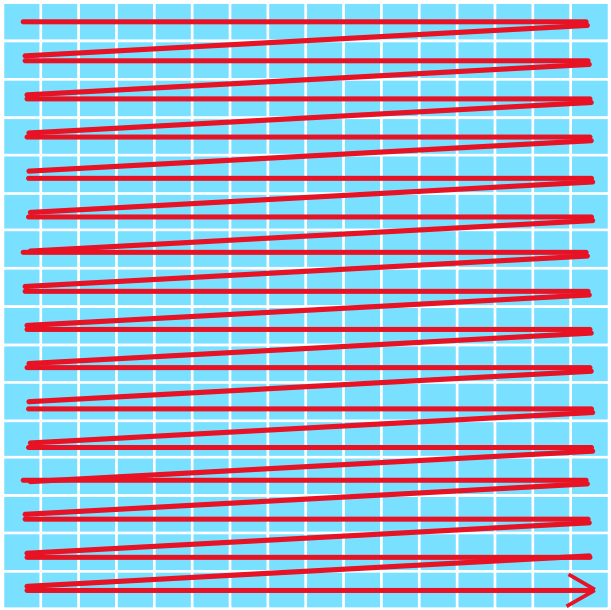
Department of Computer Science

FAU Erlangen-Nürnberg

Sommersemester 2026

Assignment 3 – Task 1

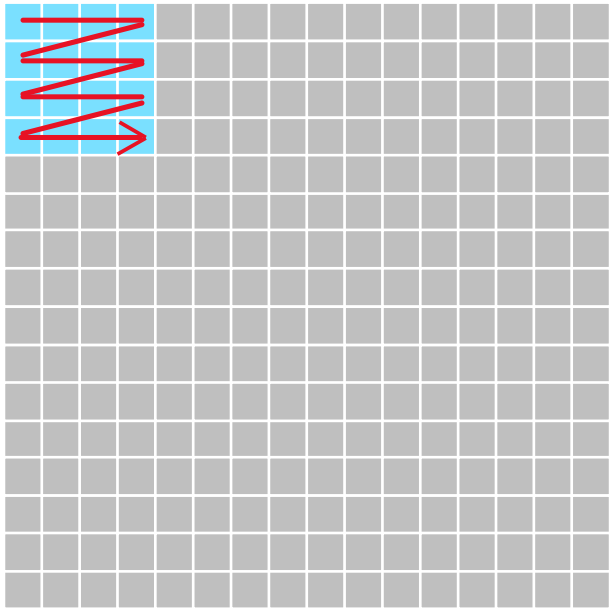
Cache optimization



```
for(r=0; r<N; ++r)
  for(c=0; c<N; ++c)
    s += m[r][c];
```

Assignment 3 – Task 1

Cache optimization



```
for(rs=0; rs<N; rs+=b)
  for(cs=0; cb<N; cs+=b)
    for(r=rs; r<rs+b; ++r)
      for(c=cs; c<cs+b; ++c)
        s += m[r][c];
```

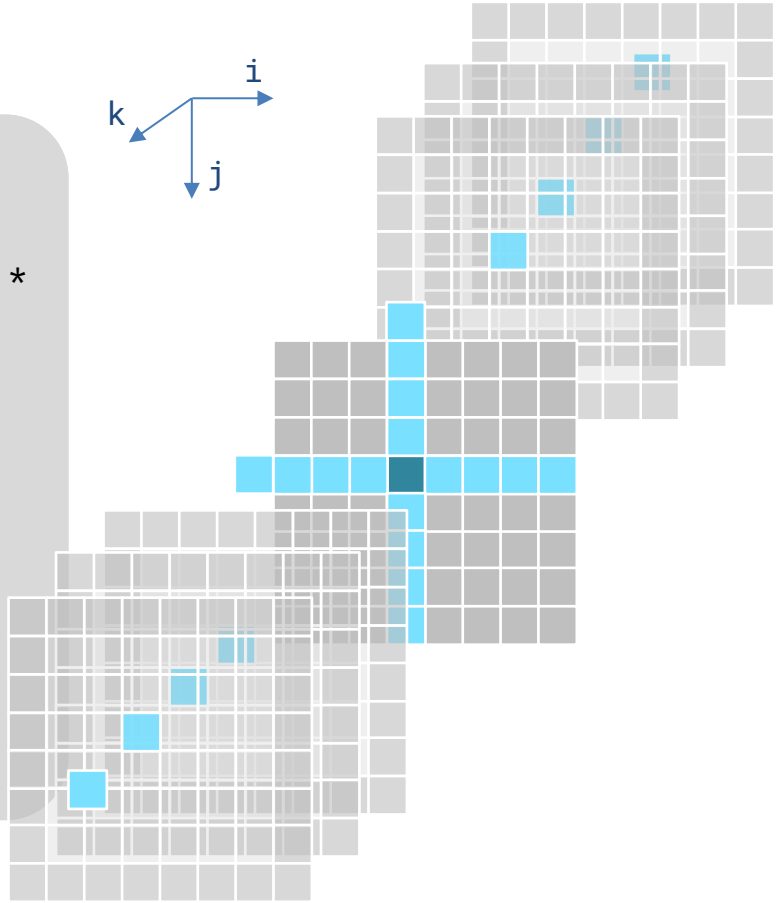
NO temporal locality

➔ No benefit from blocking (tiling)

Assignment 3 – Task 2

An intricate stencil code

```
for(int k=4; k < N-4; k++)
 for(int j=4; j < N-4; j++)
  for(int i=4; i < N-4; i++)
   U[k][j][i] = 2*V[k][j][i] - U[k][j][i] + C[k][j][i] *
    ( c0 * V[k][j][i]
     + c1 * (V[ k ][ j ][i+1] + V[ k ][ j ][i-1]
           + V[ k ][j+1][ i ] + V[ k ][j-1][ i ]
           + V[k+1][ j ][ i ] + V[k-1][ j ][ i ])
     + c2 * (V[ k ][ j ][i+2] + V[ k ][ j ][i-2]
           + V[ k ][j+2][ i ] + V[ k ][j-2][ i ]
           + V[k+2][ j ][ i ] + V[k-2][ j ][ i ])
     + c3 * (V[ k ][ j ][i+3] + V[ k ][ j ][i-3]
           + V[ k ][j+3][ i ] + V[ k ][j-3][ i ]
           + V[k+3][ j ][ i ] + V[k-3][ j ][ i ])
     + c4 * (V[ k ][ j ][i+4] + V[ k ][ j ][i-4]
           + V[ k ][j+4][ i ] + V[ k ][j-4][ i ]
           + V[k+4][ j ][ i ] + V[k-4][ j ][ i ]));
```



Assignment 3 – Task 2

An intricate stencil code

```
for(int k
 for(int
  for(int
   U[k][j                                                    j][i] *
     ( c0
      + c1
      + c2
      + c3
      + c4                                                          );
```

Microarchitecture	SandyBridge-EP	IvyBridge-EP	Haswell-EP
Shorthand	SNB	IVB	HSW
Xeon Model	E5-2680	E5-2690 v2	E5-2695 v3
Year	03/2012	09/2013	09/2014
Clock speed (fixed)	2.7 GHz	2.2 GHz	2.3 GHz
Cores/Threads	8/16	10/20	14/28
Load/Store throughput per cycle			
AVX(2)	1 LD & 1/2 ST	1 LD & 1/2 ST	2 LD & 1 ST
SSE/scalar	2 LD ‖ 1 LD & 1 ST	2 LD ‖ 1 LD & 1 ST	2 LD & 1 ST
L1 port width	2×16+1×16 B	2×16+1×16 B	2×32+1×32 B
ADD throughput	1 / cy	1 / cy	1 / cy
MUL throughput	1 / cy	1 / cy	2 / cy
FMA throughput	n/a	n/a	2 / cy
L2-L1 data bus	32 B	32 B	64 B
L3-L2 data bus	32 B	32 B	32 B
LLC size	20 MiB	25 MiB	35 MiB
Main memory	4×DDR3-1600	4×DDR3-1866	4×DDR4-2133
Peak memory BW	51.2 GB/s	51.2 GB/s	68.3 GB/s
Load-only BW	43.6 GB/s (85%)	46.1 GB/s (90%)	60.6 GB/s (89%)
T_{L3Mem} per CL	3.96 cy	3.05 cy	2.43 cy

		scalar	AVX
LOAD	27	14 cy/it	27/4 cy/it
STORE	1		
ADD/SUB	26	26 cy/it	26/4 cy/it
MULT	7		

Shift of bottleneck

Assignment 3 – Task 3

Dense MVM

Locality of access (**N**=15000):

```
for(i=0; i<N; ++i)
  for(j=0; j<N; ++j)
    c[i] += a[i][j] * b[j];
```

`a[][]` purely spatial locality (each element accessed exactly once in linear order)

```
sizeof(a) = 1.8GB
sizeof(b) = sizeof(c) = 117 KiB
```

`b[]` spatial and temporal locality:
all elements accessed in linear order,
each element reused N-1 times since b[ ] fits into L2 cache

`c[]` spatial and temporal locality:
all elements accessed in linear order,
each element reused N-1 times from L1 or register

Assignment 3 – Task 4

Loop nests

a)

```
#define N 16384
//...
s = 0.0;
for(int i=0; i<N; i++)
  for(int j=0; j<N; j++)
     s += a[j][i];
```

change idx

```
#define N 16384
//...
s = 0.0;
for(int i=0; i<N; i++)
  for(int j=0; j<N; j++)
     s += a[i][j];
```

b) $b_s = 128\ \mathrm{GB/s},\ f = 2.0\ \mathrm{GHz},\ T_l = 200\mathrm{cy}$

Transfer time: $T_{\mathrm{CL}} = 200\ \mathrm{cy} + 64\ \mathrm{byte} / 128\ \mathrm{Gbyte/s} \times 2.0\ \mathrm{Gcy/s} = 201\ \mathrm{cy}$

Effective bandwidth: $b = 64\ \mathrm{byte} / 201\ \mathrm{cy} \times 2.0\ \mathrm{Gcy/s} =$ **$0.64\ \mathrm{Gbyte/s}$**

With 128B CL: $T_{\mathrm{CL128}} = 202\mathrm{cy} \approx T_{\mathrm{CL64}}$ → **effective bandwidth doubled**

Assignment 3 – Task 4

Loop nests

c) $\mathrm{b_s} = 128\ \mathrm{GB/s},\ \mathrm{f} = 2.0\ \mathrm{GHz},\ T_l = 200\mathrm{cy},\ \mathrm{L} = 200\ \mathrm{byte}$

$$\mathrm{P} = \frac{\mathrm{T}}{\mathrm{V}/b_s} = 1 + \frac{T_l}{\mathrm{V}/b_s} = 1 + \frac{200\ \mathrm{cy}}{64\ \mathrm{byte}/64\frac{\mathrm{byte}}{\mathrm{cy}}} = \mathbf{201}$$

BUT: The CPU needs to be able to process the data!

1 scalar ADD/cy = 8 byte/cy = 16 Gbyte/s < b_s, 1 AVX ADD/cy = 64 Gbyte/s < b_s

SOLUTION: more SIMD lanes, more ADD pipelines, more cores, ...